Genetic Algorithms for Adaptive Real-Time Control in Space Systems

J. van der Zijp and A. Choudry

Center for Applied Optics
The University of Huntsville in Alabama

ABSTRACT

The U.S. Space Station planned for the 90's will comprise a large number of interacting control systems, a situation which will be too complex for humans to deal with without the aid of knowledge based systems. However, current knowledge based systems have two undesirable aspects. Brittleness: problem-solving performance degrades precipitously when the system is operating outside its intended domain, and adaptation: the ability of the knowledge base to deal with a changing environment.

Learning systems try to alleviate these problems, by organising their own knowledge base. These systems can be divided into two categories: the more traditional rule-based systems on one hand, and neural nets and genetic algorithms on the other. The latter two systems do not operate in the domain, but are 'sandwiched' between a pre- and post processing phase.

Systems based on genetic algorithms are characterized by on-the-fly improvement through a continuous contention of the available rules, while maintaining performance levels as new rules are "tried out." The most salient feature is the employment of techniques from microbiology to conquer the combinatorial explosion involved in induction of the new rules.

INTRODUCTION

Classifier systems are a domain independent way of manipulating knowledge about, say, the characteristics of a real-time control process. The classifier system is embedded into a pre- and post translation phase that relates the knowledge of the classifier system to the environment (the domain). Since it is usually not possible to incorporate all the required knowledge into the system a priori, we want the system to be able to learn.

Genetic Algorithms are used for learning as one way to control the combinatorial explosion associated with generation of plausible new rules. The Genetic Algorithm approach tends to work best when it can be applied to a domain independent knowledge representation.

As important as the generation of new rules is, the evaluation of existing rules is even more influential. Whithout it, the system will not be able to attain better performance.

CLASSIFIER SYSTEMS

As stated, a classifier system is 'sandwiched' between a separate pre- and post processing phase. The preprocessing phase will pick up the relevant information from the environment and encode this into the internal representation used by the classifier system, called a message. Likewise, the postprocessing phase converts back from the internal representation to a meaningful output signal, such as an actuator, a horn signal or whatever.

We will not discus the pre- and postprocessing phases in greater detail here, although for applications of classifier systems in real-world situations these are of course tantamount to good performance.

A classifier system comprises two parts, a message buffer and a rule base. Communication with the world outside takes place exclusively by placement or removal of messages into or from the message buffer. Also acting on the message buffer is the rule base itself. In each classification step, all messages are matched to all rules, and the matching pairs (rule,message) are allowed to write a new message into the buffer.

MESSAGE REPRESENTATION

A message is internally represented as a string of K symbols from {0,1} i.e. a binary representation. Each symbol denotes some aspect or attribute of the environment. As will be seen, this representation is very suited for genetic algorithms.

Rules in the rule base consist of two parts: a condition part and an action part. The condition part is represented as a number of strings of length K, over the alphabet {0,1,#}. The new symbol # is used here to match ANY symbol in the corresponding position in the message. For instance, condition 0101#01# will match messages 01010010, 01010011, 01011010 and 01011011. Each condition can be either asserted or negated.

The rule is said to MATCH if ALL the conditions match some message in the buffer (i.e. a match does NOT mean that one message satisfies all conditions, but rather that there is a satisfying message in the buffer for each condition). The message matching the first condition is arbitrarily called the matching message.

The action part is also represented as a string over {0,1,#}. In this case, however, the symbol # is used to denote the corresponding symbol of the message matching this rule. In other words, should a rule with action part 0101#01# match a message 11011000, then the message placed into the buffer by this rule will be 01011010.

BASIC EXECUTION CYCLE OF A CLASSIFIER SYSTEM

A classifier system goes through the following steps for each classification cycle. For legibility, this is here presented in (pseudo-) pascal style:

```
repeat
    input_messages_&_place_them_into_buffer;
    for r := 1 to no_of_rules do begin
        condition := first_condition_of_rule_r;
        message[condition] := 0;
        repeat
            message[condition] := message[condition]+1;
            if message[condition] < no_of_messages then begin
            if match(condition,message[condition]) then begin
                if condition=last_condition_of_rule_r then
                    record_match(r,condition,message[condition])
                else begin
                    condition := condition+1;
                    message[condition] := 0;
                end;
            end;
            end else
                condition := condition-1;
        until condition<0;
    end;
    for each_recorded_match do place_action_message_in_new_buffer;
    update_strength_of_matching_rules;
    replace_current_message_buffer_by_new_buffer;
    remove_messages_from_buffer_&_output_them;
until hell_freezes_over;
```

It should be noted that not all messages are recognized by the output interface as messages to be removed; only those marked as such will be output. Also note that there may be several different assignments of messages to one rule; these are all recorded.

RULE COMPETITION

Since the number of pairs (rule,message) can be extraordinarily large (with N rules, an average of C conditions per rule, and M messages there can be as much as N*C*M pairs), the number of messages that are eventually placed in the buffer are reduced by letting the matching rules enter a competition. Each matching pair produces a number called the BID, which determines the probability that the pair is allowed to place its message into the buffer. The bid is calculated as follows:

bid=factor*support(messages)*strength(rule)*specificity(rule)

where:

factor = attenuation factor;

support = sum of the bids of all messages satisfying a condition of the rule;

strength = the current "successfulness" of the rule;

specificity = 1-(number of #'s in condition)/(length of condition).

The bid computed above is an exact quantity; were we to use this quatity directly, only the strongest few rules would ever be selected in the competition and new, potentially better rules, would never have the opportunity to demonstrate their worth. Therefore, the bid used in the competition is some stochastic function of the value computed above (at the moment, normally distributed around the computed bid).

RULE ACCREDITATION

In order to allow new (low-strength) rules to acquire more strength in those situations where they are applicable, some mechanism must be devised that "gives credit to whom credit is due." That is, the well working new rules must be rewarded for providing good answers, whereas those that do not do well must sink into oblivion.

Such a mechanism has been devised by Holland. It is called the "bucket brigade." The bucket brigade algorithm works by applying the following adjustments to a rule's strength at each classification cycle:

strength(r,t+1) = strength(r,t) - bid(r,t)

strength(i,t+1) = strength(i,t) + bid(r,t)/card({R}) for i in {R}

where {R} is the set of rules responsible for the match of rule r.

In words, each rule which wins the bidding contest gets its strength reduced by the amount of its bid; each of the rules that put a message into the buffer that eventually matched one of the rule's conditions are rewarded for this by having their strength increased; they share the bid among each other.

Eventually, the rule placing a message into the buffer that is consumed by the output interface gets a reward directly by the environment.

In this way, benign "mutations" are able to gain strength. Chains of non-productive rules are eventually broken down because they will receive no credit from the environment.

RESEARCH GOALS

Our interest in classifier systems falls into two categories. On the one hand, we are concerned with the use of classifier systems for real-time adaptive control applications, and the implications of on-the-fly learning for robustness. On the other hand, we envisage some improvements to the mechanism of the classifier system itself, e.g. the guiding of the 'mutation' process by meta-rules, other kinds of genetic operators etc. Also, we want to study the effects of 'teaching' to the speed of adaptation.

REFERENCES

Holland, J. H. "Adaptation in Natural and Artificial Systems," The Univerity of Michigan Press, Ann Arbor.

Holland, J. H. "Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," Machine Learning, Vol II.

Holland, J.H., Holyoak, e.a. "Introduction to Processing of Interface Learning and Discovery," 1986 MIT Press.

Hilliard, M. R., Liepins, "A Classifier Based System for Discovering Scheduling Heuristics," 2nd Int. Conf. on Genetic Algorithms, July 1897.

Goldberg, D.E. Segrest, P., "Finite Markov Analysis of Genetic Algorithms," 2nd Int. Conf. on Genetic Algorithms.